AMENDMENTS TO THE CLAIMS

Please amend the claims as indicated in the following listing of all claims:

1. (Original) An implementation of software transactional memory that allows concurrent non-blocking access to a dynamically sizable data structure defined in shared storage managed thereby.

2. (Original) The software transactional memory implementation of claim 1, wherein the shared storage is itself dynamically sizable.

3. (Original) The software transactional memory implementation of claim 1, wherein at least some transactions that access state of the dynamically sizable data structure determine a sequence of transactional objects to access based, at least in part, on state of at least some of the transactional objects previously accessed during the same transaction.

4. (Original) The software transactional memory implementation of claim 1, further comprising:

releasing, prior to termination of a particular one of the transactions, at least some of the transactional objects previously accessed by the particular transaction.

5. (Original) The software transactional memory implementation of claim 1, wherein individual threads of a multithreaded computation that access the dynamically sizable data structure are dynamically creatable and dynamically destroyable throughout the course of the multithreaded computation.

6. (Original) The software transactional memory implementation of claim 1, wherein at least some execution sequences open transactional objects during course of a transaction and release at least some of the opened transactional objects prior to termination of the transaction.

7. (Original) The software transactional memory implementation of claim 1, wherein the concurrent non-blocking access is mediated using a single-target synchronization primitive.

8. (Original) The software transactional memory implementation of claim 7, wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

9. (Original) The software transactional memory implementation of claim 7, wherein the single-target synchronization primitive employs Load-Linked (LL) and Store-Conditional (SC) operation pair.

10. (Original) The software transactional memory implementation of claim 7, wherein the single-target of the single-target synchronization primitive includes a value and a version number encoded integrally therewith.

11. (Original) The software transactional memory implementation of claim 1, wherein the dynamically sizable data structure is implemented by a collection of transactional objects dynamically instantiable in the shared storage.

12. (Original) The software transactional memory implementation of claim 11, wherein the collection of transactional objects implements a list-oriented data structure.

13. (Original) The software transactional memory implementation of claim 11, wherein the collection of transactional objects implements a tree-oriented data structure.

14. (Original) The software transactional memory implementation of claim 11, wherein the transactional objects are implemented as objects that encapsulate objects of the dynamically sizable data structure.

15. (Original) The software transactional memory implementation of claim 1,

wherein state of the dynamically sizable data structure is accessed by a dynamically variable collection of transactions.

16. (Original) The software transactional memory implementation of claim 1, wherein the implementation is obstruction-free, though not wait-free or lock-free.

17. (Original) The software transactional memory implementation of claim 1, wherein at least some concurrently executed access operations interfere with each other; and
wherein the interfering concurrently executed access operations are retried.

18. (Original) The software transactional memory implementation of claim 1, wherein concurrently executed read access operations do not interfere with each other.

19. (Original) The software transactional memory implementation of claim 17, wherein the software transactional memory implementation does not itself guarantee that at least one of the interfering concurrently executed access operations makes progress.

20. (Original) The software transactional memory implementation of claim 1, wherein a contention management facility is employed to facilitate progress in a concurrent computation that employs the software transactional memory implementation.

21. (Original) The software transactional memory implementation of claim 20, wherein operation of the contention management facility ensures progress of the concurrent computation.

22. (Original) The software transactional memory implementation of claim 20, wherein the contention management facility is modular such that alternative contention management strategies may be employed without affecting correctness of the software transactional memory implementation.

23. (Original) The software transactional memory implementation of claim 20, wherein the contention management facility allows changes in contention management strategy during the course of a multithreaded computation that access the dynamically sizable data structure.

24. (Original) The software transactional memory implementation of claim 1, embodied as software that defines an application programming interface and which includes a functional encoding of operations concurrently executable by one or more processors to operate on state of transactional objects.

25. (Withdrawn) A method of coordinating concurrent access to a shared, dynamically sizable data structure instantiated in storage of a computer, the method comprising:

encapsulating component logical blocks of the dynamically sizable data structure within respective transactional objects, wherein at least one of the component logical blocks is introduced into the dynamically sizable data structure during the course of a concurrent computation thereon; and

for a transaction that desires write accesses to contents of a particular one of the logical blocks, aborting another active transaction, if any, that has read or write access to the particular logical block.

26. (Withdrawn) The method of claim 25, further comprising:

for a transaction that desires read access to contents of a particular one of the logical blocks, aborting another active transaction, if any that has write access to the particular logical block.

27. (Withdrawn) The method of claim 25,

wherein no active transaction may prevent it's own abortion.

28. (Withdrawn) The method of claim 25, further comprising:

cloning a private copy of the contents of the particular logical block for use by the transaction.

29. (Withdrawn) The method of claim 25, further comprising:

aborting any other active transaction, if any, that is active with respect to the particular logical block.

30. (Withdrawn) The method of claim 25, further comprising:

attempting the abort for any other active transaction that has write access to the particular logical block, the abort attempt ensuring, on completion thereof, that no other transaction has write access to the particular logical block.

31. (Withdrawn) The method of claim 25, further comprising:

attempting the abort for any other transaction that is active with respect to the particular logical block, the abort attempt ensuring, on completion thereof, that no other is active with respect to the particular logical block.

32. (Withdrawn) The method of claim 25,

wherein an abort attempt is mediated using a single-target synchronization primitive.

33. (Withdrawn) The method of claim 25, further comprising:

selecting an appropriate version of the particular logical block depending on status of a last transaction, if any, that had write access to the particular logical block.

34. (Withdrawn) The method of claim 25, further comprising:

selecting an appropriate version of the particular logical block from amongst two or more versions of contents thereof, wherein each of the versions is accessible via a current locator.

35. (Withdrawn) The method of claim 33,

wherein a first version the logical block contents is the appropriate version if status of a transaction that installed the current locator is either active or aborted; and

wherein a second version the logical block contents is the appropriate version if status of the transaction that installed the current locator is committed.

36. (Withdrawn) The method of claim 25,

wherein each transaction that opens a transactional object for write access, attempts to install a new locator using a single-target synchronization primitive to mediate concurrent attempts to update transaction status.

37. (Withdrawn) The method of claim 25,

wherein each transaction that opens a transactional object for exclusive access, attempts to install a new locator using a single-target synchronization primitive to mediate concurrent attempts to update transaction status.

38. (Withdrawn) The method of claim 36, further comprising:

on failure of the attempt to install a new locator, retrying.

39. (Withdrawn) The method of claim 25, further comprising:

committing the transaction using a single-target synchronization primitive to ensure that the committing transaction has not been aborted.

40. (Withdrawn) The method of claim 39,

wherein the single-target synchronization primitive further ensures that, on commitment of the transaction, another transaction is not active with respect to the particular logical block.

41. (Withdrawn) The method of claim 25, further comprising:

retrying an aborted transaction.

42. (Withdrawn) The method of claim 25, further comprising:

for a second particular transaction that opens a second particular one of the logical blocks for read-only access,

storing the second particular logical block and its latest committed version, and

detecting conflicts by comparing the stored version with a current committed version of the second particular logical block.

43. (Withdrawn) The method of claim 42,

wherein the stored version is represented in a read-only table that includes a counter, initially zero;

wherein the second particular transaction, and any other transaction that opens a second particular logical block for read-only access, increments the counter; and

wherein a transaction that releases a logical block decrements the respective counter and releases the block only if the counter reaches zero.

44. (Original) A computer readable medium encoding at least a portion of an implementation of software transactional memory, the encoding comprising:

a definition of a transactional object instantiable in shared memory to individually encapsulate allocatable logical blocks of a dynamically sizable data structure; and

functional encodings of a open-type operation and a commit-type operation that employ respective instances of a single-target synchronization primitive to mediate concurrent, non-blocking access to the transactional object.

45. (Original) The encoding of claim 44,

wherein the transactional object definition includes a locator that mediates access to a transaction status and versions of the corresponding encapsulated logical block; and

wherein respective instances of the single-target synchronization primitive are employed by the open-type and commit-type operations to mediate concurrent, non-blocking access to the locator and the transaction status, respectively.

46. (Original) The encoding of claim 44,

wherein instances of the transactional object are dynamically instantiable in the shared memory during the course of a multithreaded computation that accesses the dynamically sizable data structure.

47. (Original) The encoding of claim 44,

wherein individual threads of a multithreaded computation that access the dynamically sizable data structure are dynamically creatable and destroyable throughout the course of a multithreaded computation.

48. (Original) The encoding of claim 44,
wherein the software transactional memory manages a portion of the shared memory, and wherein the managed portion of is dynamically sizable.

49. (Original) The encoding of claim 44,
wherein at least some instances of the single-target synchronization primitive employ a Compare-And-Swap (CAS) operation.

50. (Original) The encoding of claim 44,
wherein at least some instances of the single-target synchronization primitive employ a Load-Linked (LL) and Store-Conditional (SC) operation pair.

51. (Original) The encoding of claim 44,
wherein the single-target of the single-target synchronization primitive includes a value and a version number encoded integrally therewith.

52. (Original) The encoding of claim 44,
embodied as a application programming interface software component combinable with program code to provide the program code with non-blocking access to a concurrent shared object.

53. (Original) A encoding of claim 44,
wherein the computer readable medium includes at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

54. (Original) A computer readable medium encoding of a dynamically sizable data structure implementation, the data structure encoding comprising:

a definition of a component logical block of the dynamically sizable data structure instantiable in shared memory; and

a functional encoding of access operations that, when executed on respective one or more processors that access the shared memory, provide concurrent non-blocking access to respective ones of the logical blocks,

the access operations invoking open-type operations of a software transactional memory implementation to open respective logical blocks of the dynamically sizable data structure;

the access operations further invoking commit-type operations of a software transactional memory implementation to commit respective logical blocks of the dynamically sizable data structure,

wherein the open-type operation and a commit-type operation employ respective instances of a single-target synchronization primitive to mediate concurrent, non-blocking access to respective transactional objects that encapsulate individual ones of the logical blocks.

55. (Original) The data structure encoding of claim 54,

wherein the concurrent non-blocking access is obstruction-free, though not wait-free or lock-free.

56. (Original) The data structure encoding of claim 54, further comprising:

a functional encoding of the open-type and commit-type operations.

57. (Original) The data structure encoding of claim 54,

wherein the single-target synchronization primitive employs a Compare-And-Swap (CAS) operation.

58. (Original) The data structure encoding of claim 54,

wherein the single-target synchronization primitive employs Load-Linked (LL) and Store-Conditional (SC) operation pair.

59. (Original) The data structure encoding of claim 54,

wherein the single-target of the single-target synchronization primitive includes a value and a version number encoded integrally therewith.

60. (Original) The data structure encoding of claim 54,
wherein the dynamically sizable data structure includes a tree-oriented data structure, individual nodes of which are encapsulated by transactional objects of the transactional memory implementation.

61. (Original) The data structure encoding of claim 54,
wherein the dynamically sizable data structure includes a list-oriented data structure, individual elements of which are encapsulated by transactional objects of the transactional memory implementation.

62. (Original) The data structure encoding of claim 54,
embodied as a software component combinable with program code to provide the program code with non-blocking access to a concurrent shared object.

63. (Original) A data structure encoding of claim 54,
embodied as a program executable to provide non-blocking access to a concurrent shared object.

64. (Original) A data structure encoding of claim 54,
wherein the computer readable medium includes at least one medium selected from the set of a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

65. (Original) An apparatus comprising:
plural processors;
one or more data stores addressable by each of the plural processors; and
means for coordinating concurrent non-blocking execution, by ones of the plural processors, of access operations that manipulate respective logical blocks of a dynamically sizable data structure, the coordinating means employing respective

instances of a single-target synchronization primitive to mediate concurrent, non-blocking open-type and commit-type operations on respective transactional objects that encapsulate individual ones of the logical blocks.

66. (Original) The apparatus of claim 65,
wherein the coordinating means tolerates non-progress of interfering executions of the access operations.

67. (Original) The apparatus of claim 65, further comprising:
means for managing contention between interfering executions of the access operations.

68. (*New*) The software transactional memory implementation of claim 1, wherein the implementation coordinates the concurrent non-blocking access by encapsulating within respective transactional objects, component logical blocks of a dynamically sizable data structure instantiated in storage of a computer, wherein at least one of the component logical blocks is introduced into the dynamically sizable data structure during the course of a concurrent computation, and, for a transaction that desires write accesses to contents of a particular one of the logical blocks, aborting another active transaction, if any, that has read or write access to the particular logical block.